

R II

Econ 5/Poli 5D Lecture 13

Announcements

- Fifth quiz due this Friday
- Final projects
- Any questions?

Continuing our question from last class: How can we detect discrimination in the labor market?

- One clever way is to use resume audit studies that send fictitious resumes to firms and look at callback rates
- Resume audit studies "hold constant" other factors that may impact hiring decisions

Today's Software/Data Skills:

- Writing logic statements and creating subsets of data

Recall arithmetic operators in R

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Exponentiation: ^ or **

Before getting to the application, we are going to learn more about logical expressions in R

Before getting to the application, we are going to learn more about logical expressions in R

- Less than: $<$ or equal to $<=$
- More than: $>$ or equal to $>=$
- Exactly equal to: $==$
- Not: $!$
- Not equal to: $!=$
- Or: $|$
- And: $\&$

In R, you can simply type logical expressions and the answer will be returned

In R, you can simply type logical expressions and the answer will be returned

```
In [1]: "Triton" != "triton"
```

```
TRUE
```

In R, you can simply type logical expressions and the answer will be returned

```
In [1]: "Triton" != "triton"
```

TRUE

```
In [2]: 5 > 3
```

TRUE

In R, you can simply type logical expressions and the answer will be returned

```
In [1]: "Triton" != "triton"
```

TRUE

```
In [2]: 5>3
```

TRUE

```
In [3]: vec <- c(1,2,3)  
sum(vec)>7
```

FALSE

Creating lists and sequences

Suppose you want a list of numbers increasing by 1

Creating lists and sequences

Suppose you want a list of numbers increasing by 1

```
In [4]: list1 <- 1:5  
list1
```

1 · 2 · 3 · 4 · 5

Creating lists and sequences

Suppose you want a list of numbers increasing by 1

```
In [4]: list1 <- 1:5  
list1
```

1 · 2 · 3 · 4 · 5

If we apply a relational statement to the list, it tests the relation for all elements

Creating lists and sequences

Suppose you want a list of numbers increasing by 1

```
In [4]: list1 <- 1:5  
list1
```

1 · 2 · 3 · 4 · 5

If we apply a relational statement to the list, it tests the relation for all elements

```
In [5]: list1 > 3
```

FALSE · FALSE · FALSE · TRUE · TRUE

Creating lists and sequences

Suppose you want a list of numbers increasing by 1

```
In [4]: list1 <- 1:5  
list1
```

1 · 2 · 3 · 4 · 5

If we apply a relational statement to the list, it tests the relation for all elements

```
In [5]: list1 > 3
```

FALSE · FALSE · FALSE · TRUE · TRUE

```
In [6]: (list1 > 1) & (list1 < 5)
```

FALSE · TRUE · TRUE · TRUE · FALSE

Class of an object

Just like variables, objects in R have different "classes"

For example, objects that are numbers are referred to as "numeric"

Class of an object

Just like variables, objects in R have different "classes"

For example, objects that are numbers are referred to as "numeric"

In [7]:

```
x <- 5  
class(x)
```

'numeric'

Class of an object

Just like variables, objects in R have different "classes"

For example, objects that are numbers are referred to as "numeric"

In [7]:

```
x <- 5  
class(x)
```

'numeric'

Words are "character" class

Class of an object

Just like variables, objects in R have different "classes"

For example, objects that are numbers are referred to as "numeric"

In [7]:

```
x <- 5  
class(x)
```

'numeric'

Words are "character" class

In [8]:

```
y <- "Hello World"  
class(y)
```

'character'

In some cases, you may need to change the class of an object.

We can **coerce** objects to be recognized as different classes

In some cases, you may need to change the class of an object.

We can **coerce** objects to be recognized as different classes

In [9]:

```
z.str <- c("1", "2", "3")  
z.num <- as.numeric(z.str)
```

In some cases, you may need to change the class of an object.

We can **coerce** objects to be recognized as different classes

In [9]:

```
z.str <- c("1", "2", "3")  
z.num <- as.numeric(z.str)
```

The `as.numeric` function is telling R to take `z.str`, which is a character class, and **coerce** it to numeric

In some cases, you may need to change the class of an object.

We can **coerce** objects to be recognized as different classes

```
In [9]: z.str <- c("1", "2", "3")
z.num <- as.numeric(z.str)
```

The `as.numeric` function is telling R to take `z.str`, which is a character class, and **coerce** it to numeric

```
In [10]: z.str
z.num
```

```
'1' '2' '3'
```

```
1 2 3
```

We can also go in the reverse direction. Turning numeric objects to characters

We can also go in the reverse direction. Turning numeric objects to characters

```
In [11]: a.num <- c(4,5,6)
a.str <- as.character(a.num)
a.str
```

```
'4' '5' '6'
```

We can also go in the reverse direction. Turning numeric objects to characters

```
In [11]: a.num <- c(4,5,6)
          a.str <- as.character(a.num)
          a.str
```

```
'4' '5' '6'
```

If R cannot **coerce** an object to the required class, it will return NA

We can also go in the reverse direction. Turning numeric objects to characters

```
In [11]: a.num <- c(4,5,6)
a.str <- as.character(a.num)
a.str
```

```
'4' · '5' · '6'
```

If R cannot **coerce** an object to the required class, it will return NA

```
In [12]: school <- c("UCLA", "UCSD")
school.num <- as.numeric(school)
school.num
```

```
Warning message in eval(expr, envir, enclos):
"NA's introduced by coercion"
```

```
<NA> · <NA>
```

Logicals

There is also a class of objects referred to logicals

Logicals

There is also a class of objects referred to logicals

```
In [13]: vec <- c(1,2,3)
```

Logicals

There is also a class of objects referred to logicals

```
In [13]: vec <- c(1,2,3)
```

```
In [14]: logical <- (vec>2)  
logical
```

FALSE · FALSE · TRUE

Logicals

There is also a class of objects referred to logicals

```
In [13]: vec <- c(1,2,3)
```

```
In [14]: logical <- (vec>2)
logical
```

```
FALSE · FALSE · TRUE
```

```
In [15]: class(logical)
```

```
'logical'
```

Sometimes in R, **coercion** may occur even if you do not intend it.

Sometimes in R, **coercion** may occur even if you do not intend it.

In [16]:

```
vec
```

```
1 · 2 · 3
```

Sometimes in R, **coercion** may occur even if you do not intend it.

In [16]:

```
vec
```

```
1 · 2 · 3
```

Let's change the first entry of `vec` to the word "missing"

Sometimes in R, **coercion** may occur even if you do not intend it.

In [16]:

```
vec
```

```
1 · 2 · 3
```

Let's change the first entry of `vec` to the word "missing"

In [17]:

```
vec[1] <- "missing"  
vec  
class(vec)
```

```
'missing' · '2' · '3'
```

```
'character'
```

Sometimes in R, **coercion** may occur even if you do not intend it.

```
In [16]: vec
```

```
1 2 3
```

Let's change the first entry of vec to the word "missing"

```
In [17]: vec[1] <- "missing"  
vec  
class(vec)
```

```
'missing' '2' '3'
```

```
'character'
```

R has **coerced** the remaining entries of vec to characters

A Matrix

You can also create matrices in R, which you can think of as a collection of vectors

For example:

A Matrix

You can also create matrices in R, which you can think of as a collection of vectors

For example:

In [18]:

```
vec1 <- c(1,2)
vec2 <- c(3,4)

matrix <- as.matrix(cbind(vec1,vec2))
```

```
matrix
```

A matrix: 2 ×
2 of type dbl

vec1	vec2
1	3
2	4

Matrix Notation

To retrieve an element of a matrix, we can use brackets `[]` which are used for subsetting in R

`matrix[i, j]` refers to the entry in the row and column

So `matrix[1, 2]` refers to the entry in the first row, second column

Matrix Notation

To retrieve an element of a matrix, we can use brackets `[]` which are used for subsetting in R

`matrix[i,j]` refers to the entry in the i^{th} row and j^{th} column

So `matrix[1,2]` refers to the entry in the first row, second column

In [19]:

```
matrix
matrix[1,2]
```

A matrix: 2 ×
2 of type dbl

vec1	vec2
1	3
2	4

vec2: 3

Matrix Notation

To retrieve an element of a matrix, we can use brackets `[]` which are used for subsetting in R

`matrix[i,j]` refers to the entry in the i^{th} row and j^{th} column

So `matrix[1,2]` refers to the entry in the first row, second column

In [19]:

```
matrix
matrix[1,2]
```

A matrix: 2 ×
2 of type dbl

vec1	vec2
1	3
2	4

vec2: 3

In [20]:

```
matrix[2,2]
```

vec2: 4

You can also use brackets to subset entire rows of a matrix

You can also use brackets to subset entire rows of a matrix

In [21]:

```
row1<- matrix[1,]  
as.vector(row1)
```

1.3

You can also use brackets to subset entire rows of a matrix

In [21]:

```
row1<- matrix[1,]  
as.vector(row1)
```

1 3

The **1** indicates we want the first row. The fact that we left the second entry blank indicates we want to retrieve **all columns**

We can do the same idea for retrieve columns of a matrix

We can do the same idea for retrieve columns of a matrix

In [22]:

```
col2 <- matrix[,2]  
as.vector(col2)
```

3 · 4

We can do the same idea for retrieve columns of a matrix

In [22]:

```
col2 <- matrix[,2]  
as.vector(col2)
```

3 · 4

Since we left the first entry blank, R knows we want all the rows. Since we put **2** for the second entry it returns only the second column

Practice

Try to create the following matrix

[,1]	[,2]	[,3]	
[1,]	1	2	3
[2,]	4	5	6

And then use indexing to retrieve (1) the second column and then (2) the first row

In [23]:

```
mat <- matrix(1:6, ncol = 3, nrow = 2, byrow = T)
mat
mat[,2]
mat[1,]
```

A matrix:

2 × 3 of

type int

1	2	3
4	5	6

2 · 5

1 · 2 · 3

Dataframes work in a similar way in R (with a few modifications). We can subset the data using brackets `[]`

Dataframes work in a similar way in R (with a few modifications). We can subset the data using brackets `[]`

Let's load our data and remind ourselves what is in the data

Dataframes work in a similar way in R (with a few modifications). We can subset the data using brackets `[]`

Let's load our data and remind ourselves what is in the data

In [24]:

```
setwd('/Users/Brian/Dropbox/Grad School/Sixth Year/Econ:Poli 5/Lectures/Week 8/data')
resume <- read.csv("./resume.csv")
head(resume)
```

A data.frame: 6 × 5

	X	firstname	sex	race	call
	<int>	<fct>	<fct>	<fct>	<int>
1	1	Allison	female	white	0
2	2	Kristen	female	white	0
3	3	Lakisha	female	black	0
4	4	Latonya	female	black	0
5	5	Carrie	female	white	0
6	6	Jay	male	white	0

We can think our data frame as a large matrix.

The number of rows is equal to the number of observations

The number of columns is equal to the number of variables

Therefore, we can think of it as a matrix of dimension `[n rows, n columns]`

The function `dim()` retrieves the dimension of the matrix

```
In [25]: ### Using the dim function
```

```
In [25]: ### Using the dim function
```

```
In [26]: dim(resume)
```

```
4870 · 5
```

```
In [25]: ### Using the dim function
```

```
In [26]: dim(resume)
```

```
4870 · 5
```

Interpretation -- There are 4870 observations and 5 variables

We can retrieve the same information with the functions `nrow()` and `ncol()`

```
In [27]: nrow(resume)
```

```
4870
```

```
In [28]: ncol(resume)
```

```
5
```

Imagine we want to retrieve the vector of names (the second column of the data frame)

Imagine we want to retrieve the vector of names (the second column of the data frame)

```
In [29]: namevec <- resume[,2]
```

Imagine we want to retrieve the vector of names (the second column of the data frame)

```
In [29]: namevec <- resume[,2]
```

```
In [30]: head(namevec)
```

Allison · Kristen · Lakisha · Latonya · Carrie · Jay

► **Levels:**

Imagine we want to retrieve the vector of names (the second column of the data frame)

```
In [29]: namevec <- resume[,2]
```

```
In [30]: head(namevec)
```

Allison · Kristen · Lakisha · Latonya · Carrie · Jay

► **Levels:**

For dataframes, recall we can reference a certain column by its variable name as well

Imagine we want to retrieve the vector of names (the second column of the data frame)

```
In [29]: namevec <- resume[,2]
```

```
In [30]: head(namevec)
```

Allison · Kristen · Lakisha · Latonya · Carrie · Jay

► **Levels:**

For dataframes, recall we can reference a certain column by its variable name as well

```
In [31]: namevec2 <- resume$firstname  
head(namevec2)
```

Allison · Kristen · Lakisha · Latonya · Carrie · Jay

► **Levels:**

To subset our data, we will combine our knowledge of brackets with the `which()` command, but there are many ways to subset your data in R (the `subset()` command is also popular)

To use `which()`, you need to put a logical statement in the parentheses

For example, `which(resume$race=="black")` returns a list of all the rows in which the logical statement is true

To subset our data, we will combine our knowledge of brackets with the `which()` command, but there are many ways to subset your data in R (the `subset()` command is also popular)

To use `which()`, you need to put a logical statement in the parentheses

For example, `which(resume$race=="black")` returns a list of all the rows in which the logical statement is true

```
In [32]: flag_black <- which(resume$race=="black")
         head(flag_black)
```

```
3 4 8 9 10 11
```

To subset our data, we will combine our knowledge of brackets with the `which()` command, but there are many ways to subset your data in R (the `subset()` command is also popular)

To use `which()`, you need to put a logical statement in the parentheses

For example, `which(resume$race=="black")` returns a list of all the rows in which the logical statement is true

```
In [32]: flag_black <- which(resume$race=="black")
         head(flag_black)
```

```
3 · 4 · 8 · 9 · 10 · 11
```

```
In [33]: flag_white <- which(resume$race=="white")
         head(flag_white)
```

```
1 · 2 · 5 · 6 · 7 · 12
```

Now we can combine the `which()` command (which gives us a list of rows) that satisfy a given statement with our knowledge of subsetting

Now we can combine the `which()` command (which gives us a list of rows) that satisfy a given statement with our knowledge of subsetting

```
In [34]: # Black Applicants data frame  
df_b <- resume[which(resume$race=="black"),]  
  
# White Applicants data frame  
df_w <- resume[which(resume$race=="white"),]
```

In [35]:

```
head(df_b)
```

A data.frame: 6 × 5

	X	firstname	sex	race	call
	<int>	<fct>	<fct>	<fct>	<int>
3	3	Lakisha	female	black	0
4	4	Latonya	female	black	0
8	8	Kenya	female	black	0
9	9	Latonya	female	black	0
10	10	Tyrone	male	black	0
11	11	Aisha	female	black	0

In [35]:

```
head(df_b)
```

A data.frame: 6 × 5

	X	firstname	sex	race	call
	<int>	<fct>	<fct>	<fct>	<int>
3	3	Lakisha	female	black	0
4	4	Latonya	female	black	0
8	8	Kenya	female	black	0
9	9	Latonya	female	black	0
10	10	Tyrone	male	black	0
11	11	Aisha	female	black	0

In [36]:

```
head(df_w)
```

A data.frame: 6 × 5

	X	firstname	sex	race	call
	<int>	<fct>	<fct>	<fct>	<int>
1	1	Allison	female	white	0
2	2	Kristen	female	white	0
5	5	Carrie	female	white	0
6	6	Jay	male	white	0
7	7	Jill	female	white	0
12	12	Allison	female	white	0

Now let's look at the average callback by race

Now let's look at the average callback by race

In [37]:

```
# Black applicant callback rate  
mean(df_b$call)  
  
#white applicant callback rate  
mean(df_w$call)  
  
# difference  
mean(df_b$call) - mean(df_w$call)
```

0.064476386036961

0.0965092402464066

-0.0320328542094456

Now let's look at the average callback by race

In [37]:

```
# Black applicant callback rate  
mean(df_b$call)  
  
#white applicant callback rate  
mean(df_w$call)  
  
# difference  
mean(df_b$call) - mean(df_w$call)
```

0.064476386036961

0.0965092402464066

-0.0320328542094456

Interpretation: Black applicants about 3.2 percentage points less likely to receive a callback, despite these applications being similar on all other dimensions

We can also combine logical expressions to create further splits of the data

We can also combine logical expressions to create further splits of the data

In [38]:

```
df_b_f <- resume[which(resume$race=="black" & resume$sex=="female"),]  
df_b_m <- resume[which(resume$race=="black" & resume$sex=="male"),]  
  
df_w_f <- resume[which(resume$race=="white" & resume$sex=="female"),]  
df_w_m <- resume[which(resume$race=="white" & resume$sex=="male"),]
```

In [39]:

```
# callback rate for Black and Female  
mean(df_b_f$call)  
  
# callback rate for White and Female  
mean(df_w_f$call)  
  
# difference  
mean(df_b_f$call) - mean(df_w_f$call)
```

0.0662778366914104

0.0989247311827957

-0.0326468944913853

In [39]:

```
# callback rate for Black and Female  
mean(df_b_f$call)  
  
# callback rate for White and Female  
mean(df_w_f$call)  
  
# difference  
mean(df_b_f$call) - mean(df_w_f$call)
```

0.0662778366914104

0.0989247311827957

-0.0326468944913853

Slightly higher callback rates for female applicants, but overall disparity by race is very similar

In [40]:

```
# callback rate for Black and Female  
mean(df_b_m$call)  
  
# callback rate for White and Female  
mean(df_w_m$call)  
  
# difference between call back rates  
mean(df_b_m$call) - mean(df_w_m$call)
```

0.058287795992714

0.088695652173913

-0.030407856181199

In [40]:

```
# callback rate for Black and Female  
mean(df_b_m$call)  
  
# callback rate for White and Female  
mean(df_w_m$call)  
  
# difference between call back rates  
mean(df_b_m$call) - mean(df_w_m$call)
```

0.058287795992714

0.088695652173913

-0.030407856181199

Slightly lower callback rates for male applicants relative to female applicants, but overall disparity by race is very similar

We could also run a regression to look at the same relationship

We could also run a regression to look at the same relationship

```
In [41]: reg1 <- lm(call ~ race, data = resume)
reg1
```

Call:

```
lm(formula = call ~ race, data = resume)
```

Coefficients:

(Intercept)	racewhite
0.06448	0.03203

In [42]:

```
summary(reg1)
```

Call:

```
lm(formula = call ~ race, data = resume)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-0.09651	-0.09651	-0.06448	-0.06448	0.93552

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.064476	0.005505	11.713	< 2e-16 ***
racewhite	0.032033	0.007785	4.115	3.94e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2716 on 4868 degrees of freedom

Multiple R-squared: 0.003466, Adjusted R-squared: 0.003261

F-statistic: 16.93 on 1 and 4868 DF, p-value: 3.941e-05

In [43]:

```
summary(lm(call ~ race, data = resume[which(resume$sex=="female"),]))
```

Call:

```
lm(formula = call ~ race, data = resume[which(resume$sex == "female"),  
    ])
```

Residuals:

	Min	1Q	Median	3Q	Max
	-0.09892	-0.09892	-0.06628	-0.06628	0.93372

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.066278	0.006325	10.478	< 2e-16 ***
racewhite	0.032647	0.008977	3.637	0.00028 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2747 on 3744 degrees of freedom

Multiple R-squared: 0.00352, Adjusted R-squared: 0.003254

F-statistic: 13.23 on 1 and 3744 DF, p-value: 0.0002796

Today's Application: Resume Audit studies provide convincing evidence of discrimination in the labor market

- This idea has been replicated for many different types of discrimination
- For example, gender, age, ethnicity, among others

Today's Software/Data Skills:

- Learn about object classes
- What is a matrix and how to subset elements of a matrix
- How to subset a dataframe